Module 2:

Introduction to Streamlit

307307 – BI Methods and Models

What is Streamlit?

- Streamlit is a Python framework for building interactive web applications.
- It is widely used for data visualization, dashboards, and AI demos.
- You write Python code only — Streamlit handles the web interface automatically.
- No need for HTML, CSS, or JavaScript.
- Ideal for prototyping LLM-powered apps quickly.

Installing Streamlit

pip install streamlit

streamlit hello

Opens a demo at http://localhost:8501

Shows the Streamlit layout, widgets, and charts in action

Your First App

Create a file called app.py:

```
import streamlit as st
st.title("My First Streamlit App")
st.write("Hello, Streamlit!")
```

Run it:

```
streamlit run app.py
```

Explanation:
st.title() adds a large title at the top of the page.
st.write() displays text, numbers, dataframes, or any Python object in a readable format.

Display Elements

Text and Formatting

```
st.header("Section Header")        # Adds a smaller header
st.subheader("Subsection")         # Adds a subheader
st.text("Plain text output")       # Displays plain text
st.markdown("**Markdown works too**")  # Renders markdown-formatted text
st.code("print('Python code block')", language='python')  # Shows code nicely formatted
```

Explanation:
These commands help you organize content visually and make apps easier to read.

Display Data

```
import pandas as pd
data = pd.DataFrame({'x':[1,2,3],'y':[10,20,30]})


st.dataframe(data)    # Interactive table
st.table(data)        # Static table
st.line_chart(data)   # Simple line chart
st.bar_chart(data)    # Simple bar chart
```

Explanation:
Streamlit automatically detects the data structure and renders appropriate visualizations.

Widgets: User Inputs

```
name = st.text_input("Enter your name")
age = st.slider("Select your age", 0, 100, 25)


if st.button("Submit"):
    st.write(f"Hello {name}, you are {age} years old!")
```

Explanation:

st.text_input() adds a text box.

st.slider() adds an interactive slider.

st.button() triggers code when clicked.

These widgets allow the user to provide inputs to the app.

Common Widgets

Other useful widgets include:

st.checkbox() – true/false input

st.radio() – choose one from a list

st.selectbox() – dropdown menu

st.multiselect() – select multiple options

st.date_input() – choose a date

st.file_uploader() – upload a file

Each widget updates automatically when the user interacts with it.

Maintenance State

By default, Streamlit reruns the script whenever a widget changes.
Use st.session_state to store information between interactions.

```
if 'count' not in st.session_state:
    st.session_state.count = 0


if st.button("Increment"):
    st.session_state.count += 1


st.write("Counter:", st.session_state.count)
```

Explanation:
st.session_state keeps variables (like chat history) persistent across user actions.

Organizing Layouts

Columns

```
col1, col2 = st.columns(2)
col1.write("Left column")
col2.write("Right column")
```

Tabs

```
tab1, tab2 = st.tabs(["Data", "Chart"])
with tab1:
    st.write("Show data here")
with tab2:
    st.line_chart(data)
```

Sidebar

```
option = st.sidebar.selectbox("Choose an option", ["A", "B", "C"])
st.write("You chose:", option)
```

Explanation:

These features help structure your app and make it more user-friendly.

File Uploads, Downloads, and Caching

```
uploaded = st.file_uploader("Upload CSV", type="csv")
if uploaded:
    df = pd.read_csv(uploaded)
    st.write(df)


st.download_button("Download CSV", df.to_csv(), "data.csv")
```

Caching

```
@st.cache_data
def load_data():
    return pd.read_csv("large_file.csv")
```

Explanation:
Caching prevents recomputing expensive functions, improving performance.

Why Streamlit for LLM Apps

- Simple interface for chat-based input and output
- Easy management of conversation history
- Quick integration with APIs (Gemini, OpenAI, etc.)
- Local testing and cloud deployment are straightforward
- Ideal for creating AI chatbots and text-based assistants

Use Case: Simulating ChatGPT with Gemini

You'll build a simple **chat interface** using:

1. **Streamlit** for UI
2. **Gemini API** for responses
3. **Session state** for memory
4. **Outcome:** A working chatbot web app running locally or online.

Setup

1. Install dependencies:

pip install streamlit google-generativeai python-dotenv

2. Get an API key:
🔗 https://makersuite.google.com/app/apikey

Chat App Code (Basic Version)

```
import streamlit as st
import google.generativeai as genai

# --- Set up Gemini API key directly ---
genai.configure(api_key="YOUR_GEMINI_API_KEY_HERE")
# --- Initialize model ---
model = genai.GenerativeModel("gemini-1.5-flash")
# --- Streamlit UI setup ---
st.set_page_config(page_title="Chat with Gemini", page_icon="💬")
st.title("Chat with Gemini")
# --- Maintain conversation state ---
if "messages" not in st.session_state:
    st.session_state.messages = []
# --- Display chat history ---
for msg in st.session_state.messages:
    role = "You" if msg["role"] == "user" else "Gemini"
    st.markdown(f"**{role}:** {msg['content']}")
# --- Handle user input ---
prompt = st.chat_input("Type your message...")
```

Chat App Code (Basic Version)

```
if prompt:
    # Add user message
    st.session_state.messages.append({"role": "user", "content": prompt})
    st.markdown(f"**You:** {prompt}")


    # Send prompt to Gemini
    with st.spinner("Gemini is thinking..."):
        chat = model.start_chat(history=st.session_state.messages)
        response = chat.send_message(prompt)
        reply = response.text


    # Add and display model response
    st.session_state.messages.append({"role": "model", "content": reply})
    st.markdown(f"**Gemini:** {reply}")
```

Explanation of Key Functions

- st.chat_input() creates a text box at the bottom of the app.
- model.start_chat() initializes a Gemini conversation with history.
- chat.send_message() sends the prompt to the model.
- Responses are displayed using st.markdown().
- Chat history is stored in st.session_state.messages.

Running the App

Run locally:

streamlit run app.py

Then open in browser:
➡️ http://localhost:8501

You'll see:

- Chat history
- Input box
- Responses from Gemini

How It Works

- User enters a message.
- Streamlit captures it and stores it in session state.
- Gemini API receives the conversation context.
- Gemini returns a generated response.
- Both messages appear in the chat window.
- The process repeats interactively.

Student Challenge

- Choose your own **use case** (e.g., summarize text, generate code, tutor bot)
- Build it in Streamlit
- Connect to any LLM API (Gemini, OpenAI, Claude, etc.)
- Deploy it publicly on Streamlit Cloud
- Share your app link!